Architecting On-Chip Interconnects for Stacked 3D STT-RAM Caches in CMPs

Asit K. Mishra Xiangyu Dong Guangyu Sun
Yuan Xie N. Vijaykrishnan Chita R. Das
Department of Computer Science and Engineering
The Pennsylvania State University
{amishra, xydong, gsun, yuanxie, vijay, das}@cse.psu.edu

ABSTRACT

Emerging memory technologies such as STT-RAM, PCRAM, and resistive RAM are being explored as potential replacements to existing on-chip caches or main memories for future multi-core architectures. This is due to the many attractive features these memory technologies posses: high density, low leakage, and non-volatility. However, the latency and energy overhead associated with the write operations of these emerging memories has become a major obstacle in their adoption. Previous works have proposed various circuit and architectural level solutions to mitigate the write overhead. In this paper, we study the integration of STT-RAM in a 3D multi-core environment and propose solutions at the on-chip network level to circumvent the write overhead problem in the cache architecture with STT-RAM technology. Our scheme is based on the observation that instead of staggering requests to a write-busy STT-RAM bank, the network should schedule requests to other idle cache banks for effectively hiding the latency. Thus, we prioritize cache accesses to the idle banks by delaying accesses to the STT-RAM cache banks that are currently serving long latency write requests. Through a detailed characterization of the cache access patterns of 42 applications, we propose an efficient mechanism to facilitate such delayed writes to cache banks by (a) accurately estimating the busy time of each cache bank through logical partitioning of the cache layer and (b) prioritizing packets in a router requesting accesses to idle banks. Evaluations on a 3D architecture, consisting of 64 cores and 64 STT-RAM cache banks, show that our proposed approach provides 14% average IPC improvement for multi-threaded benchmarks, 19% instruction throughput benefits for multi-programmed workloads, and 6% latency reduction compared to a recently proposed write buffering mechanism.

Categories and Subject Descriptors

B.3.2 [**Hardware**]: Memory Structures – cache memories; C.1.2 [**Computer Systems Organization**]: Multiprocessors; Interconnection architectures

General Terms

Design, Experimentation, Measurement, Performance.

ISCA'11, June 4–8, 2011, San Jose, California, USA.

1. INTRODUCTION

Emerging memory technologies such as Magnetic RAM (MRAM), Phase-change RAM (PCRAM), and Resistive RAM (RRAM) are being explored as potential alternatives to existing memories such as SRAM and DRAM. Among these technologies, Spin-Torque Transfer RAM (STT-RAM) [8, 10] combines the speed of SRAM, the density of DRAM, and the non-volatility of Flash memory, with excellent scalability. Furthermore, it has been shown that with 3D stacking [1, 24], STT-RAM can be integrated with conventional CMOS logic [4]. Thus, STT-RAM is potentially attractive to replace the traditional on-chip SRAM [7, 12, 16, 17, 19, 23, 26], with benefits such as higher density and lower leakage compared to the traditional SRAM-based cache architectures.

Even though the STT-RAM based cache architecture has many advantages, it suffers from longer write latency and higher write energy consumption when compared to SRAM based architecture. The write energy and duration are higher since, to write a "0" or "1" into an STT-RAM cell, a strong current is necessary to force the storage node (Magnetic Tunnel Junctions, or MTJ) to reverse the magnetic direction [8, 25]. The latency and energy overhead associated with the write operations of these emerging memories has become a major obstacle in their widespread adoption.

To address the latency/energy overhead associated with the write operations in STT-RAM, researchers have proposed various mitigation techniques at both the circuit level and architectural level. For example, circuit-level approaches such as eliminating redundant bit-writes [26] and data inverting [9] have been proposed to reduce the write energy; architectural techniques such as read-preemptive write-buffer design, write termination [26], and hybrid cache/memory architecture [17, 19] can also help mitigate the latency and energy overhead in STT-RAM.

In this paper, we study the integration of STT-RAM in a 3D multi-core environment and propose solutions at the on-chip network level to circumvent the write overhead problem. Our scheme is based on the observation that instead of using the network for sending subsequent requests to a write-busy STT-RAM bank, where they will be queued until the bank is free, the network should serve requests to idle cache banks to hide the write latency. Thus, we *selectively grant network resources* to the cache requests for accessing idle STT-RAM banks by prioritizing these requests over requests to busy banks. We examine the cache access distributions of different commercial and server workloads and observe that on an average, it is possible to delay 17% (up to 27%) of accesses to an STT-RAM bank for giving priority to other cache accesses, and thereby hide the memory latency.

Despite this simple observation and our early analysis showing the potential benefits of selective prioritization of requests in a network, determining which cache banks are busy and which banks

are idle in a distributed environment is non-trivial. Thus, the main body of this work presents a mechanism for accurately predicting the busy/idle status of a cache bank a few hops away from the destination and a request prioritization mechanism in the routers for effectively utilizing the idle cache banks. We use a two-layer stacked architecture of cores and STT-RAMs in different layers to demonstrate our idea using both multi-threaded and multi-programmed workloads with a diverse set of 42 applications. The main *contributions* of our paper are the following:

STT-RAM aware router arbitration: First, we show why the simple round-robin arbitration is not optimal for using STT-RAM and then elaborate our proposal of packet re-ordering in a 3D network (Section 3). Next, we describe a scheme for facilitating prioritization in a 3D network by (a) partitioning the STT-RAM layer into a number of logical regions, (b) restricting the path diversity for accurate prediction of a cache bank status, and (c) estimating the busy duration of STT-RAMs using three novel schemes: simple scheme (SS), regional congestion aware scheme (RCA), and window based scheme (WB).

Quantifying the benefits of STT-RAM aware router arbitration: In Section 4, we present experimental results with a two-layer 64-core and 64-STT-RAM configuration and show that our proposed approach can lead to an average 14% improvement in IPC (with multi-threaded benchmarks), 19% improvement in instruction throughput (with multiprogrammed workloads), and 54% reduction in energy compared to an SRAM-based cache implementation. Additionally, we find that logically partitioning the cache layer into more regions helps our schemes by having a fine-grained control of the packets and accurately estimating the congestion that improves performance further. We find that re-ordering requests from a router two hops away from the destination STT-RAM bank and sub-dividing the cache layer into eight logical regions gives the best performance results (19% IPC improvement) for our proposed prioritization schemes. We also conduct several sensitivity analysis to examine the scalability and inflection points of our proposed approach.

Benefits of our scheme over prior proposals: We show that our scheme is better (can provide 6% additional reduction in latency) when compared to a recently proposed write buffering technique [19] that uses write-buffers in every STT-RAM bank and employs read-preemption at the cache bank level. The novelty of the proposed network-based solution is that it does not require additional resources such as extra buffers at each STT-RAM bank for write-buffering. The already available VCs can be used to implement our scheme. To the best of our knowledge, this is the first work that shows how the on-chip network facilitates STT-RAM based cache architectures for CMPs.

2. BACKGROUND

This section gives a brief background on STT-RAM, on-chip networks, and 3D integration technologies.

STT-RAM: Unlike the traditional SRAM and DRAM technologies that use electric charges to store information, STT-RAM uses Magnetic Tunnel Junctions (MTJs) for binary storage. As shown in Figure 1, an MTJ contains two ferromagnetic layers and one tunnel barrier layer (MgO). The direction of one ferromagnetic layer (called reference layer) is fixed, while the direction of the other layer (called free layer) can be changed by forcing a driving current. The relative magnetization direction of the free layer and the reference layer determines the resistance of MTJ. If two ferromagnetic layers have the same directions, the resistance of MTJ is low, indicating a "0" state and vice-versa for a "1" state.

MTJ is the storage element of STT-RAM and a memory cell can

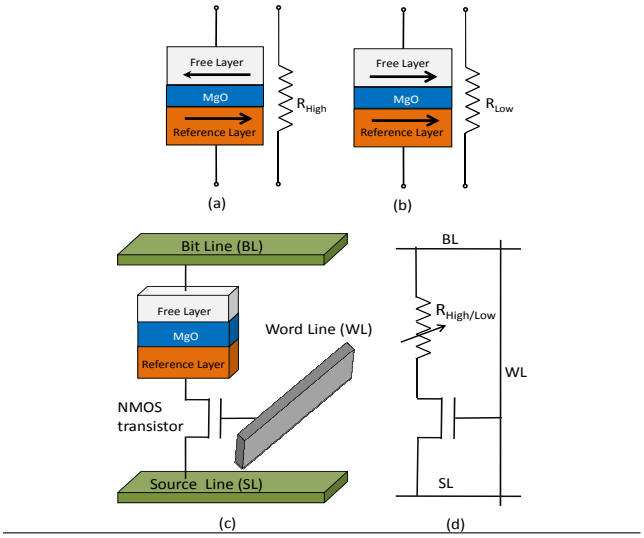


Figure 1: MTJ and STT-RAM cell (a) Anti-parallel (high resistance), indicating "1" state (b) Parallel (low resistance), indicating "0" state (c) STT-RAM structural view (d) STT-RAM schematic.

be designed using a *one-transistor-one-MTJ* ("1T1J") structure [8, 10]. As illustrated in Figure 1, each MTJ is connected in series with an NMOS. The gate of the NMOS is connected to the word line (WL), and the NMOS is turned on if its connected MTJ needs to be accessed during read or write operations. The source of the NMOS is connected to the source line (SL), and the free ferromagnetic layer is connected to the bit line (BL). The STT-RAM data read mechanism uses sense amplifiers to sense the voltage difference caused by the resistance difference of MTJ in a "0" and "1" status. The read latency of STT-RAM can be as short as the SRAM read latency and the read energy of STT-RAM is also comparable to the SRAM read energy. However, the write duration as well as the write energy of STT-RAM are significantly higher than that of an SRAM write access [4, 19].

Network-on-Chip (NoC) architectures: A packet-based NoC provides a scalable interconnection fabric for connecting the processor nodes, the on-chip shared cache banks and the on-chip memory controllers [2]. On-chip routers and links constitute this scalable communication backbone. A generic NoC router has P input and P output channels/ports; typically P = 5 for a 2D mesh, one from each cardinal direction and one from the local node. The main components of a router are the routing computation unit (RC), virtual channel arbitration unit (VA), switch arbitration unit (SA), and a crossbar to connect the input and output ports. The RC unit is responsible for determining the next router based on the packet address and the virtual channel (VC) in the next router for each packet. The VA unit arbitrates amongst all packets requesting access to the same VCs and decides the winners. The SA unit arbitrates amongst all VCs requesting access to the crossbar and grants permission to the winning packets/flits. The winners are then able to traverse the crossbar and be placed on the output links. State-of-the-art wormhole switched NoCs devote two to four pipeline stages to these components [15] and typically employ dimension-ordered routing (e.g. X-Y routing) to route packets in the network.

The two arbitration stages (VA and SA), where a router must choose one packet/flit among several packets/flits competing for either a common output VC or a crossbar output port, play a major role in selecting packets for transmission. Current router im-

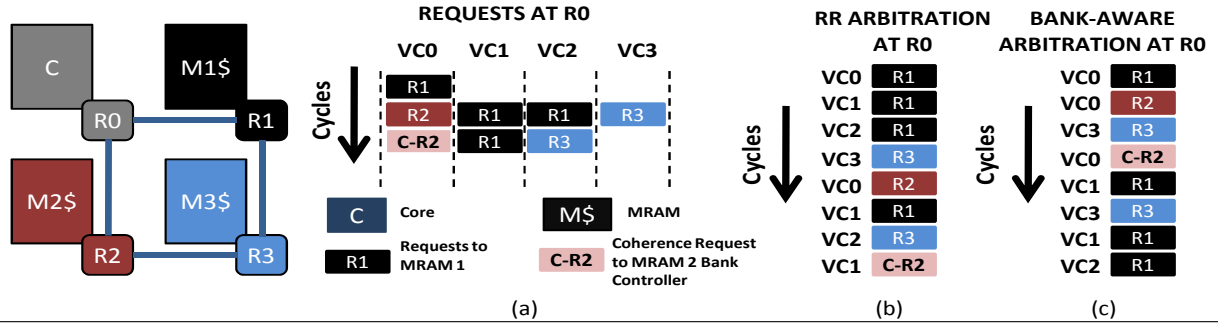


Figure 2: (a) Example request sequence at R0 in a 2x2 mesh topology; arbitration sequence using (b) simple round-robin arbiter and (c) an STT-RAM bank aware arbiter.

plementations use simple, local arbitration policies such as round robin (RR) to decide which packet should be scheduled next. We propose to modify these local and architecturally oblivious arbiters to prioritize packets for hiding the long STT-RAM write latency.

3D integration technology: 3D integration is a technology that stacks multiple active silicon dies vertically, and connects them through die or wafer bonding. The multiple dies communicate through through-silicon vias (TSVs) [22]. Among various 3D architectures, stacking cache or memory chips directly on top of a 2D multi-core chip [1, 11, 13] has gained popularity because such an architecture provides fast and high memory bandwidth between the core layer and memory layer. In this paper, we exploit the 3D stacking for putting the STT-RAM based cache banks in one layer of the CMP configuration.

3. STT-RAM AWARE NOC DESIGN

3.1 A case for STT-RAM aware router arbitration

To motivate the importance of an STT-RAM aware router arbitration, we present an example illustrated in Figure 2. In this example, the network consists of 4 routers connected in a 2x2 mesh, where router R0 is connected to a processing core (C) and the other routers (R1-R3) are connected to STT-RAM cache banks[1]. Consider the case where R0 receives multiple write requests from C over time as shown in Figure 2(a). The resulting arbitration sequence at R0 that employs the traditional RR policy routes multiple requests to R1 before forwarding requests to R2 and R3 (see Figure 2(b)). Since writes to STT-RAMs have a long latency, subsequent accesses within a short duration of the first write to the STT-RAM module connected to R1, are queued at the STT-RAM module interface (possibly at the network interface). This STT-RAM oblivious arbitration, not only leads to degraded performance (since banks connected to R2 and R3 remain idle), but also leads to unwanted network congestion at R1 (which might affect other flows going through R1 in a larger network).

In contrast, with an STT-RAM aware arbitration, R0 can prioritize requests to STT-RAM modules (R2 and R3) over a request to R1. This can be done by being cognizant of the scenario that the module connected to R1 would be busy with the write request that was sent to it recently. Such a scheme can yield performance improvement by (a) prioritizing requests to idle banks and (b) shifting the buffering of requests to busy STT-RAM modules from the module interface to the network router buffers. An example schedule resulting from the STT-RAM aware arbitration is shown in Figure 2(c). This prioritization scheme is specifically beneficial for STT-RAM structures (and possibly for other memory technologies with long bank access times) but not attractive for conventional SRAM cache banks. This is because, SRAM write latencies are typically of the order of router and network latencies. Thus, delaying a write-request to a busy SRAM bank would not overlap with the service time in the bank and hence, may hurt performance.

[1]For brevity, STT-RAM is annotated as *MRAM* in all the figures and plots.

3.2 Our proposal: Re-ordering accesses to STT-RAM banks

Since write-latencies for STT-RAMs (33 cycles; see Table 2) are about 11 times larger than that of router hop latency (3 cycles), we prioritize requests to banks other than the one recently forwarded with a write request. The key intuition behind this is - the bank to which a write request has been sent would be busy servicing the request and sending more requests to this bank would only queue them. Thus, not all requests become equally critical from a network stand-point and a router can schedule requests prioritizing few over the others. Another aspect of this re-ordering and selective prioritization can be used to *prioritize packets to memory controllers and coherence traffic*. For example, if the destination of all cache request packets in a particular router is to an already busy cache bank, coherence packets and packets destined to memory controllers can be prioritized (see Figure 2(c)) to boost performance.

Next, we need to address two critical questions: (1) *How* long should a packet be delayed? and (2) *Where* should a packet be re-ordered? i.e. how far from its destination? Ideally, a packet to a busy bank should be delayed such that it arrives at the busy bank immediately after the previous write request has been serviced. Thus, the network and queuing delays of the subsequent (delayed) packets should overlap with the service time of the first packet to avoid performance degradation. The ability to detect the busy duration of STT-RAM banks in the network is critical to achieving this overlap. If we were to prioritize and re-order requests in a router far away from the requests' destination, estimating the congestion in the network and busy time of a destination cache bank becomes difficult. In contrast, if we were to do the re-ordering of packets from a router very close to the destination cache bank, then this particular router would not have many requests to re-order and hence, would be forced to route a packet to a busy cache bank (only to be later queued at that bank). After an extensive sensitivity analysis, we choose to selectively delay request packets in a router two hops away from the destined busy STT-RAM bank. Qualitatively, there are two reasons behind this choice: (1) Estimating busy time of a destination bank that is two hops away from the current router is relatively easy (shown later in Section 4.3)) and, (2) it gives us

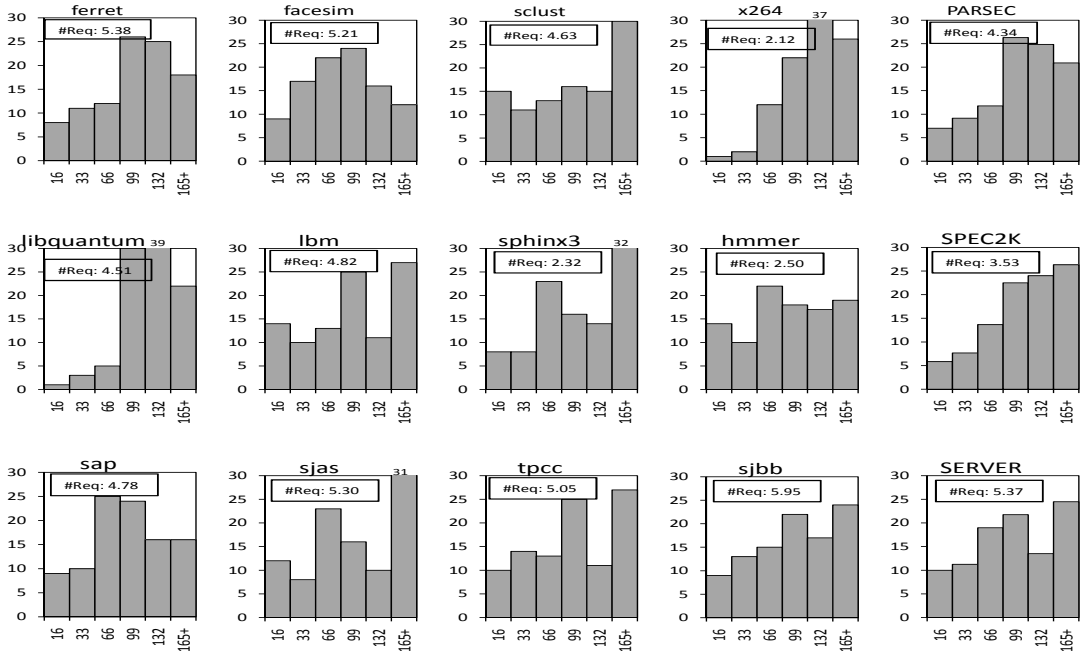


Figure 3: Plots showing the distribution of consecutive accesses to STT-RAM banks in different applications following a write access(the last column shows the average across the whole benchmark suite). The horizontal axis represents the access latencies in cycles and the vertical axis represents the percentage of accesses. The inset in each plot denotes the average number of request-packets in a router in the cache layer to two hops away STT-RAM locations.

significant opportunity to prioritize coherence, memory-controller destined traffic, and any other packets destined to routers more than two hops away.

Two factors that decide the success of our re-ordering/prioritizing scheme are: (1) How separated in time (cycles) are two consecutive accesses to a cache bank. For example, if the inter-arrival gap of successive writes to the same bank is much larger than 33 cycles, there is not enough scope for our approach. (2) Number of different cache banks to which requests are buffered in a router indicates the potential of the re-ordering scheme (and also dictates our decision in choosing two hops as the distance for re-ordering). For example, if all requests in a router are destined to a particular bank, then re-ordering those requests is not useful. The following subsection analyzes these two factors.

3.3 Cache access distribution

For different applications, we analyzed the distribution (in cycles) of consecutive accesses to STT-RAM banks and the average number of request packets buffered in a router in the cache layer, whose destination is two hops away from the current router. In this analysis, we have 64 cores in one layer and 64 STT-RAM banks in the other layer (see Section 4.1 for details on our simulation testbed). Figure 3 shows the result of this analysis, where the vertical axis represents the percentage of all accesses to a cache bank following a write access to that bank and the horizontal axis corresponds to the latency in cycles. The graphs are plotted with 33 cycle intervals (except the first bin), with the leftmost bar indicating the percentage of accesses that are separated by $0 <$ latency < 16 cycles after a write request. The rightmost bar represents the percentage of all accesses that are separated by more than 165 cycles.

This plot quantitatively shows the burstiness in applications. For `ferret`, around 8% of the accesses occur within 16 cycles after a write request is initiated to a cache bank and 10% of the accesses within 33 cycles after a write request is initiated. Considering that write service times in STT-RAMs last 33 cycles (Table 2), all these subsequent requests inevitably get queued in the network-interface of the router or the bank controller before getting serviced. Requests that are separated by at least 33 cycles (after a write request to a cache bank), are not queued and can directly proceed to be serviced. These requests are represented in the 66, 99, 132 and 165+ bins in the histograms shown. The figure also shows that not all applications are bursty. For instance, requests in `x264` are spread out and the percentage of requests following a write request to a bank that can potentially get queued (i.e. in bins 16 and 33) is only 4%. However, across all applications we analyzed, on an average, 17% of requests (up to 27% for write-intensive and burtsy applications) are always queued behind a write operation. Figure 3 also shows the number of request packets in a router in the cache layer whose destination is exactly two hops away and that follows a write access request. We find that, almost always there are about *3 requests* in a router following a write packet, which if scheduled soon after the write request would end up getting queued in the STT-RAM banks. These are the request packets that can potentially be *delayed* to *hide* the long latency write operations of STT-RAMs.

3.4 Facilitating prioritization

To prioritize requests to idle banks and delay requests to busy banks, a router should be able to know which banks in its vicinity are idle and which banks are busy. One way of achieving this, is to have a global network that transmits this information to every router in the network on a cycle-by-cycle basis. This approach would, however, be highly expensive for resource and power constrained NoCs. An alternate approach, could be to route all packets destined to a particular cache bank through a particular router in the network - a serialization point. This serialization point can serve as a secondary source for the cache bank to which all other nodes

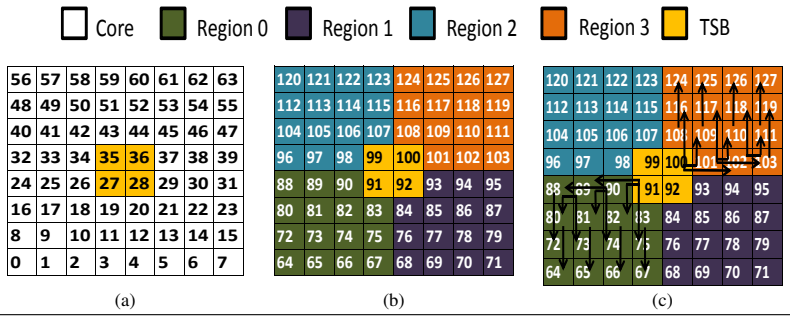


Figure 4: Two layers of the 3D CMP: (a) Core layer (b) Cache layer (c) Cache layer showing the child nodes of parent nodes.

wanting to communicate with the cache bank would send their requests. However, this is not the case in current implementations of 3D NoC. In a typical 3D network, each router in the core layer is connected to a router in the cache layer below. When using a deterministic routing algorithm like Z-X-Y or X-Y-Z routing, owing to the path-diversity, each cache may receive requests from various cores along different routes. For example, consider the illustration shown in Figure 4 with the core layer being on top of the cache layer and core 0 connected to router 0, core 1 connected to router 1 and so on. If core 0 wants to send a request to cache node 64, the request packet would be routed vertically downwards. If core 63 wants to send a request to cache node 0, with Z-X-Y routing, the request packet is routed vertically to router 127, followed by X-direction routing from router 127 to router 120 and then Y-direction routing to router 0. Since these two routes never overlap, there is no single point in the network, where we can re-order requests if cache bank 64 were to be busy.

We introduce a novel scheme to provide such a serialization point in the network. Our scheme involves (1) dividing the cache layer into logical regions and (2) limiting the path-diversity in the 3D NoC.

Partition the cache layer for reducing path-diversity: We partition the cache layer of 64 cache banks into a few logical regions - 4 to demonstrate the concept. We designate one vertical through-silicon bus (TSB)[2] in each region through which all cores can communicate their request packets with any cache bank in the corresponding region in the cache layer. Figure 4 shows the logical partition and the 4 TSBs connecting one router in the core layer to one router in the cache layer. Use of X-Y routing together with these TSBs, serializes packets to routers in each region. Hence, a packet routed from the core layer to the cache layer is first routed using X-Y routing to a particular router in the core layer, followed by a TSB traversal in the vertical direction to a router in the cache region. Finally, the packet follows X-Y routing again in the cache layer to the destination cache bank. No such path restriction is imposed when communicating between the cache layer to the core layer i.e. all 64 TSBs can be used in this case. Also, coherence traffic is not constrained to go through only the 4 TSBs, and can use all the 64 TSBs.

Note that the innermost corner router in a region does not form a serialization point for *all* the traffic in that region. Each router manages traffic for all two hops away routers in the region. We call the router that manages traffic for the two hops away destination STT-RAM bank as a *parent node* and a router connected to the STT-RAM bank two hops away from the parent node as a *child node*. Thus, few routers in each region serve as the parent nodes, where the prioritization of core-to-cache packets occur and these parent nodes have a predicted estimate of the busy duration of the two hops away banks from them (Section 3.5 describes this prediction schemes). As an example, in Figure 4, router 91 manages traffic to cache bank 75, 82 and 89 and router 90 manages traffic to cache banks 74, 81 and 88. Since all packets in the cache layer use X-Y routing, using 4 TSBs and partitioning the cache layer into logical regions help each parent router to estimate the busy duration of banks two hops away from it. The innermost corner three nodes in each region that lie in the center of the network (ex. nodes 83, 90 and 91 of region 0) are managed by the region-TSB node vertically above in the core layer (i.e. node 27).

Clearly, restricting the routes from the core layer to the cache layer would increase the hop count and hurt performance. To reduce the performance penalty of increased hop count, instead of sending one flit at a time through the core to cache layer TSBs, we combine few flits and transmit them simultaneously. Our design is similar to the XShare technique [3], where an NoC router combines two small flits (coherence and header flits) and sends them over the link to the next router. However, in our scheme, whenever possible, we combine two 128b data flits and transfer them simultaneously over the high density TSB (256b in our design).

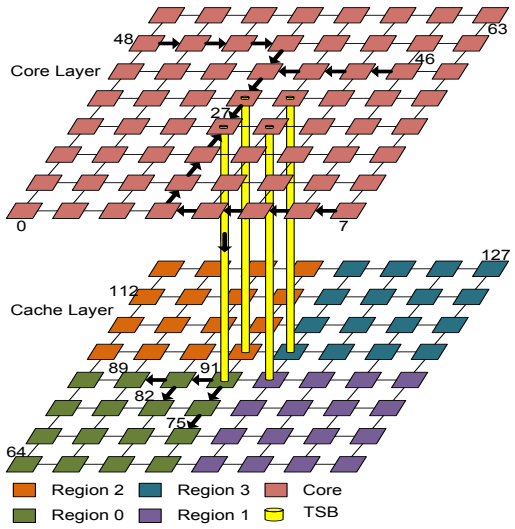


Figure 5: Proposed 3D architecture with cores in the top layer and STT-RAM banks (partitioned into 4 logical regions) in the bottom layer. The bold arrows show the route taken by requests from core to cache bank.

Putting it all together: Figure 5 shows our resulting 3D NoC de-

[2]We call a collection of TSVs to be a TSB

sign. In this design, all communications from the core to the cache layer occur through the 4 high density TSBs (256b), while communication from the cache layer to the core layer can use all of the 64 128b TSBs. This figure shows the paths taken for requests generated at cores 7, 46, and 48 to communicate with cache banks 89, 82, and 75, respectively. For all these requests, since the destination cache node lie in region 0, they are first routed using X-Y routing to node 27 in the same layer, followed by a vertical TSB transaction, and finally being buffered in the secondary source router 91. The router 91 now sees all the requests and since it is the only router through which all requests to routers 75,82, and 99 pass, it has an estimate of which of these cache banks are currently busy or idle, and thus, can selectively prioritize these requests. The next section describes how each parent router estimates the busy duration of its child nodes so that it can delay a request packet to the child node when the child node is busy.

3.5 Estimation of busy duration

Restricting the path diversity by allowing only 4 TSBs to be used when communicating from the cores to the caches and using X-Y routing in the cache regions, helps each parent node estimate the busy duration of its child nodes. This is because, each child node only receives requests from its parent nodes. Also, ideally a parent node should delay a request to its child node so that as soon as the child node is done servicing a request, another request arrives from its parent node. The latency of a packet from a parent router to the two hops away destination consists of router delay, link traversal delay, and delay due to congestion. Now, since the destination is two hops away, there is one intermediate router with 2 cycles delay (we assume a 2-stage router) and 2 cycles link traversal (1 cycle each) delay. The only unknown component is the congestion at the intermediate and destination nodes. Thus, each parent node should delay a request by 4 cycles + estimated congestion cycles + write service time in the STT-RAM bank (= 33 cycles) *following* a write-request. The delay is kept track using counters and busy-bits are maintained for each child node to identify a busy cache bank. We use three heuristics for this congestion estimation:

Simplistic Scheme (SS): In the simplistic scheme, the parent node delays a request packet to a *busy* cache bank for 33 cycles following a write-request by ignoring the congestion. While delaying packets for this duration, the parent node prioritizes packets destined to other parent nodes, coherence traffic and traffic to memory controllers. Clearly, in this scheme, since congestion is not modeled, a packet is not sufficiently delayed when congestion at the destination bank is significant and arrives at its destination only to be queued.

Regional Congestion Aware (RCA) scheme: In the RCA scheme, we use information from neighboring nodes to estimate congestion. This scheme is based on Grot et al.'s [6] scheme, where a coarse view of regional congestion can be obtained by aggregating congestion estimates from neighboring routers. In our RCA based scheme, an aggregation module resides at each network interface and the inputs to the aggregation module come from downstream routers and the local router's congestion estimate. Aggregation logic then combines the two congestion values, potentially weighting one value differently than the other, and feeds the result to a propagation module that propagates the congestion estimate of a router to its neighbors. To estimate the local congestion, a router uses buffer utilization in the port along which congestion estimates are propagated. Similar to [6], we weigh the local and neighboring congestion information equally. The RCA scheme requires additional wires for propagating the congestion estimates among neighbors and based on [6], we use 8-bit extra wires between each node. Among the three schemes we use for congestion estimate, RCA provides the best estimate albeit at the cost of additional wires.

Window Based (WB) scheme: The third scheme is a window based scheme that does not require any back wiring overheads to pass congestion information from child nodes to the parent. In this scheme, for every N packets, the parent node tags a packet with a B-bit time stamp and starts a B-bit counter before despatching it to the destination node. The destination (child) node, after receiving the tagged packet, sends an ACK packet together with the time stamp it received back to the parent node. The B-bit counter is updated every cycle until an ACK packet with the B-bit time stamp is received by the parent node from the child node. After receiving the ACK packet, the parent node estimates the congestion as half of the difference between the current time and the received time stamp. This scheme is similar to the window based scheme used in TCP/IP protocol except that the size of the window is just 1. In our case, we chose N=100 and B=8 and the time stamp is appended with the header flit (counter saturation and roll-over is accounted by using additional minimal logic). A header usually carries source-destination information and is typically shorter (64b) than data flits (128b) and appending an 8-bit time stamp to the header flit does not require additional wires nor introduces any significant overhead. Congestion can vary with program phases and our analysis shows that updating the congestion information every 100 packets provides reasonably accurate congestion estimates. The overhead in WB scheme is that of maintaining B-bit counters in each router and communicating ACK messages. Our synthesized implementation of the counter scheme implemented in Verilog shows minimal gate count increase in each router and hence, the WB scheme is simpler to *implement* in the network.

4. EXPERIMENTAL EVALUATION

4.1 Experimental platform

Architecture model: We evaluate our techniques using a trace-driven, cycle-accurate x86 CMP simulator. Table 1 provides the configuration of our baseline case which contains 64 cores on the top layer and 64 L2 cache banks on the bottom layer (each layer is laid out as an 8x8 mesh NoC). Our CPU model consists of an out-of-order core. The memory instructions are modeled through the detailed memory hierarchy and network model mentioned in Table 1. The memory hierarchy uses a two-level directory-based MESI cache coherence protocol. A 3D network connects the cores, L2 cache banks, and memory controllers. We assume 3.5 mm^2 area for each core at 32 nm technology based on a scaled version of Sun's Rock core [20]. The STT-RAM die, placed directly below the core die, is partitioned into 64 banks. Based on estimates from CACTI 6.0 [14], an 1 MB SRAM cache bank and 4MB STT-RAM bank and associated router/controller have an area roughly equal to the area of one core as shown in Table 2. The 3D TSV model parameter is based on [19].

Router model: Each router uses a state-of-the-art two-stage microarchitecture based on [15]. We use the deterministic X-Y-Z routing algorithm, finite input buffering, wormhole switching, and virtual-channel flow control. A message (packet) consists of eight 128-bit flits and one header flit. The router was implemented in Verilog and synthesized using the Synopsys Design Compiler to obtain performance and area numbers for a 32nm technology node. The resulting design operates at 3GHz at 1V. Since power characterization for our target library was not available, we incorporated dynamic and leakage power numbers from Orion [21] in our simulator for detailed power analysis of the network.

STT-RAM model: To model the STT-RAM based cache memory,

Table 1: Baseline processor, cache, memory and network configuration

Processor Pipeline	3 GHz processor, 128-entry instruction window
Fetch/Exec/Commit width	2 instructions per cycle in each core; only 1 can be a memory operation
L1 Caches	32 KB per-core (private), 4-way set associative, 128B block size, 2-cycle latency, write-back, split I/D caches, 32 MSHRs
L2 Caches	1MB banks, shared, 16-way set associative, 128B block size, 3-cycle bank latency, 32 MSHRs
Main Memory	4GB DRAM, up to 16 outstanding requests for each processor, 320 cycle access, 4 on-chip Memory Controllers
Network Router	2-stage wormhole switched, virtual channel flow control, 6 VCŠs per Port, 5 flit buffer depth, 8 flits per Data Packet, 1 flit per address packet
Network Topology	3D network, each layer is an 8x8 mesh, each node in layer 1 has a router, processor, private L1 cache, each node in layer 2 has an L2 bank 4 memory controllers (1 at each corner node in layer 2), 128b bi-directional links in each layer; 128b TSBs except region TSBs, which are 256b.

Table 2: SRAM and STT-RAM comparison at 32nm

	Area	Read Energy	Write Energy	Leakage Pow at 80°C	Read lat	Write lat	Read@3GHz	Write@3GHz
1MB SRAM	3.03 mm^2	0.168 nJ	0.168 nJ	444.6 mW	0.702 ns	0.702 ns	3 cycles	3 cycles
4MB STT-RAM	3.39 mm^2	0.278 nJ	0.765 nJ	190.5 mW	0.880 ns	10.67 ns	3 cycles	33 cycles

we use the technology parameters reported in [4,8,10,19]. Because the critical current will increase dramatically when write pulse duration is shorter than 10ns [8], we confine the write pulse duration within 10ns. Based on the analysis in [25], we scaled from the related 0.18μm STT-RAM prototype chip [8] to obtain the current amplitude we need to switch the STT-RAM cells from one state to another in 32nm technology. The parameters we used in our experiments are shown in Table 2

Benchmarks: We use 42 applications consisting of *multi-threaded* PARSEC, four commercial workloads and, *multi-programmed* SPEC 2006 benchmarks. For each application, we simulate at least 50 million instructions (3200 million instructions across the 64 processors). Table 3 shows the characterization of our application suite. The reported parameters are for the applications running alone on the baseline CMP system with the L2 composed of STT-RAM cells.

Design scenarios: To show the benefits of our proposal, we analyze the following six design scenarios in our experimental platform:

- **SRAM-64TSB:** This is our baseline scheme, where all L2 cache banks are composed of SRAM cells and where there is no restriction on path diversity, i.e. all 64 TSBs are used for communication from the core layer to the cache layer.
- **STT-RAM-64TSB:** This scheme is similar to SRAM-64TSB except that now all L2 banks are composed of STT-RAM cells.
- **STT-RAM-4TSB:** This scheme is similar to STT-RAM-64TSB scheme, except that only 4 TSBs are used when communicating between the core layer and the cache layer, i.e. request packets. All 64 TSBs are still used when communicating from the cache layer to the core layer (response packets). We analyze this scheme to quantify the performance degradation solely due to restrictions in path-diversity.
- **STT-RAM-4TSB-SS:** In this scheme, all L2 cache banks are composed of STT-RAM cells, only 4 TSBs are used for request packets from the core to cache layer and we employ selective prioritization of requests to idle banks. This scheme employs the Simplistic Scheme (SS) for estimating congestion in the network.
- **STT-RAM-4TSB-RCA:** This scheme is the same as STT-RAM-4TSB-SS with the exception that we employ RCA to estimate congestion in the network.
- **STT-RAM-4TSB-WB:** This scheme is similar to STT-RAM-4TSB-SS with the exception that we employ the window based (WB) scheme to estimate congestion.

Performance evaluation metrics: Our primary performance evaluation metrics are IPC (for multi-threaded benchmarks), instruction throughput, and weighted speedup metric (for multi-programmed workloads). We define instruction throughput to be the sum total of the number of instructions committed per cycle (IPC) in the entire CMP (Eq. (1)). The weighted speedup metric [18], sums up the slowdown experienced by each application in a workload, compared to its stand alone run under the same configuration (Eq. (2)), and is widely regarded as a *system throughput* metric [5]. For the case studies using the multi-programmed workloads, we also evaluate *application-level* fairness using the maximum application slowdown metric (Eq. (3)). For multi-programmed workloads (SPEC 2006 benchmarks), we run 64 copies of the same application on 64 cores (except for the case studies, where we use workload combinations) and report improvements for the slowest copy. Similarly, with multi-threaded applications, the improvements reported are with the slowest threads.

$$Instruction\ throughput = \sum_i IPC_i \quad \textbf{(1)}$$

$$Weighted\ speedup = \sum_i \frac{IPC_i^{shared}}{IPC_i^{alone}} \quad \textbf{(2)}$$

$$Max.\ Slowdown = max_i \left(\frac{IPC_i^{alone}}{IPC_i^{shared}} \right) \quad \textbf{(3)}$$

4.2 Experimental results

Performance comparison: Figure 6 shows the IPC and instruction throughput improvements for a subset of applications we evaluated (the average, however, is taken across all applications). For clarity in the figures, all the reported numbers are normalized to the IPC (or instruction throughput) with the baseline design (SRAM-64TSB). We find that, with STT-RAM-64TSB, benchmarks having a high percentage of reads to the L2 benefit in performance. This benefit comes from the 4x increase in the size of an L2 bank when SRAM is simply replaced by STT-RAM. However, for benchmarks that have a high percentage of writes to the L2, simply replacing an SRAM bank with an STT-RAM bank, hurts performance. This, in spite of the 4x capacity increase of the STT-RAM banks, is because of the high write latency of the STT-RAM cells. In our analysis, all server benchmarks, six benchmarks from the PARSEC suite, and seven benchmarks from the SPEC-2006 suite show performance degradation when SRAM banks are replaced by STT-RAM banks. With STT-RAM-4TSB, due to restriction on path diversity for request packets, performance further degrades when compared to that of STT-RAM-64TSB scheme. However, for L2 read intensive benchmarks, performance is still better compared to SRAM-64TSB (solely due to 4x capacity increase with STT-RAM banks). With STT-RAM-64TSB, we see 6% IPC degradation across all server benchmarks and only 1% performance improvement with PARSEC benchmarks. Additionally, with the SPEC 2006 workloads, there is just 7% improvement in instruction throughput. Clearly, simply

Table 3: Benchmark Table. l1mpki: L1 misses per 1000 instructions, l2mpki: L2 misses per 1000 instructions, l2wpki: L2 writes per 1000 instructions l2rpki: L2 reads per 1000 instructions, Bursty: (High/Low) based on latency between 2 consecutive requests to a L2 bank.

No.	Benchmark	l1mpki	l2mpki	l2wpki	l2rpki	Bursty	No.	Benchmark	l1mpki	l2mpki	l2wpki	l2rpki	Bursty
1	tpcc	51.47	6.06	40.9	10.57	High	22	lbm	36.49	18.88	30.76	5.73	High
2	sjas	41.54	4.48	35.06	6.48	High	23	hmmer	34.36	3.31	12.5	21.86	High
3	sap	29.91	3.84	23.57	6.15	High	24	xalancbmk	29.7	21.07	3.02	26.68	Low
4	sjbb	25.52	7.01	19.42	6.09	High	25	leslie	26.09	18.06	7.65	18.45	Low
5	streamcluster(sclust)	29.28	8.34	15.23	14.05	High	26	sphinx	25.55	10.91	0.97	24.58	High
6	vips	13.51	8.07	6.61	6.89	High	27	gobmk	22.81	8.68	8.02	14.79	High
7	canneal	12.8	5.47	6.52	6.27	Low	28	astar	20.03	4.21	6.11	13.92	Low
8	dedup	12.8	4.59	7.42	5.36	High	29	bzip2	19.29	10.02	2.66	16.63	High
9	ferret	11.62	9.16	6.39	5.22	Low	30	milc	19.12	18.67	0.05	19.06	Low
10	facesim	10.62	6.82	6.15	4.46	Low	31	libquantum	12.5	12.5	0	12.5	Low
11	swaptions (swptns)	5.47	6.35	2.46	3.00	Low	32	omnetpp	10.92	10.15	0.25	10.67	Low
12	blackscholes (bscls)	5.29	3.73	2.80	2.48	Low	33	povray	9.63	7.86	0.88	8.75	High
13	bodytrack (bdtrk)	5.62	5.71	2.81	2.81	Low	34	gcc	9.39	8.51	0.06	9.34	High
14	raytrace (rtrce)	5.65	4.98	3.62	2.03	Low	35	namd	8.85	5.11	0.65	8.19	High
15	x264	4.17	4.62	1.87	2.29	Low	36	gromacs	5.36	3.18	0.32	5.05	High
16	fluidanimate (fldnmt)	4.89	4.41	2.68	2.2	Low	37	tonto	5.26	0.55	3.52	1.74	High
17	freqmine (frqmn)	2.29	3.96	1.31	0.98	Low	38	h264	4.81	2.74	2.03	2.78	High
18	gemsfdtd	104.04	94.62	0.8	103.23	Low	39	dealII	4.41	2.36	0.35	4.06	High
19	mcf	99.81	64.47	5.45	94.37	Low	40	sjeng	3.93	2	0.92	3.01	Low
20	soplex	48.54	16.88	19.59	28.95	Low	41	wrf	1.8	0.75	0.88	0.92	Low
21	cactus	43.81	15.64	18.65	25.16	Low	42	calculix	0.33	0.23	0.03	0.29	Low

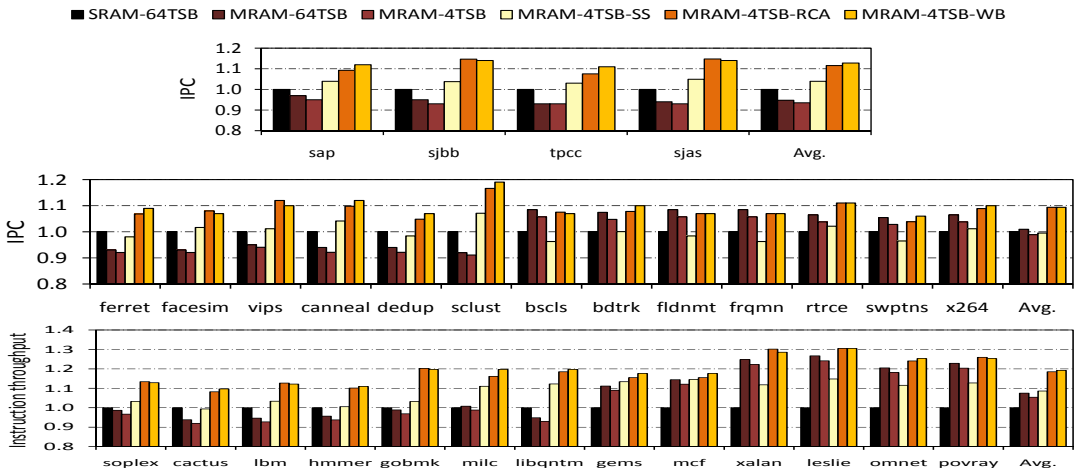


Figure 6: System throughput of the benchmarks normalized to the SRAM-64TSB case (from top to bottom: IPC for server and PARSEC benchmarks and instruction throughput with multi-programmed SPEC-2006) .

replacing SRAM with STT-RAM cells does not translate to significant performance improvement in spite of the 4x capacity increase.

We find that, if a benchmark is not very bursty, then the STT-RAM-4TSB-SS scheme, in spite of using only 4 TSBs for sending requests from the core to cache layer, on an average improves performance. However, if an application is bursty, STT-RAM-4TSB-SS scheme is not able to manage the requests to busy and idle banks well enough due to the lack of congestion estimation among parent and child nodes in this scheme. On an average, the STT-RAM-4TSB-SS scheme improves performance by 2.5% across the benchmarks evaluated. If an application is read intensive, STT-RAM-4TSB-SS shows performance improvement over simply using STT-RAM-64TSB, since, even though STT-RAM-4TSB-SS has no built-in congestion estimation, selectively delaying the accesses to a bank following a write request to the same bank in the application helps improve performance. Performance is substantially improved when we employ STT-RAM-4TSB-RCA and STT-RAM-4TSB-WB schemes. These schemes by virtue of better congestion estimation and prioritization of requests to idle banks, improve the performance significantly over SRAM-64TSB. We find STT-RAM-4TSB-RCA to be slightly (1%) better than the STT-RAM-4TSB-WB scheme. Since, the RCA scheme has overheads of back-wiring, our proposed WB scheme works well across all applications with performance close to the RCA scheme. On an average, we observed 14% and 9% IPC improvement with server and PARSEC benchmarks, respectively, and 19% improvement in instruction throughput with SPEC-2006 benchmarks.

Network packet latency: To further investigate the reason behind performance benefits with our proposal, we analyzed the reduction in network packet latency for a few benchmarks. Figure 7 shows the results of this analysis, where we have broken down a packet's latency into network latency (router latency and network congestion latency) and queuing latency at memory banks. We observe that queuing latency constitutes a significant portion of a packet's latency and this component worsens (average 8%) by replacing the SRAM banks with STT-RAM banks due to the increased write latency. However, by prioritizing requests to idle banks and estimating congestion accurately within a buffered request's vicinity, all of

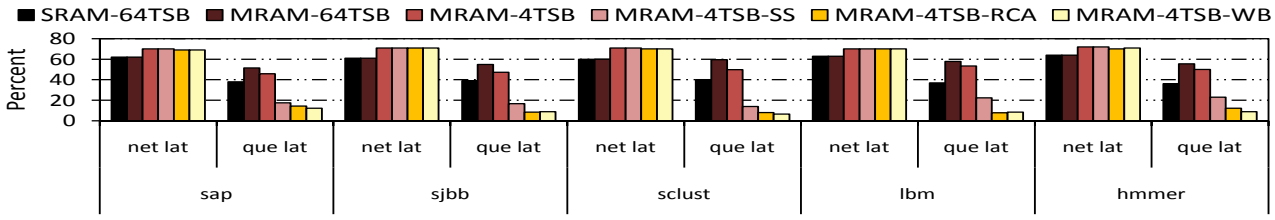


Figure 7: Packet latency breakdown into network latency (net lat) and queuing latency at memory banks (queue lat). SRAM-64TSB are exact percentages. All other values are normalized to that of SRAM-64TSB.

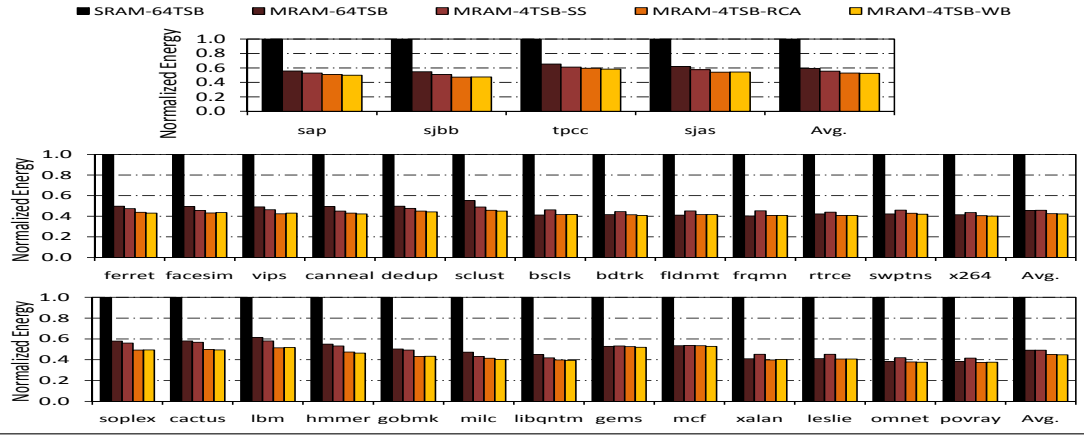


Figure 8: Energy of the benchmarks normalized to that of SRAM-64TSB.

the three schemes reduce the queueing delay component up to 35% in the network.

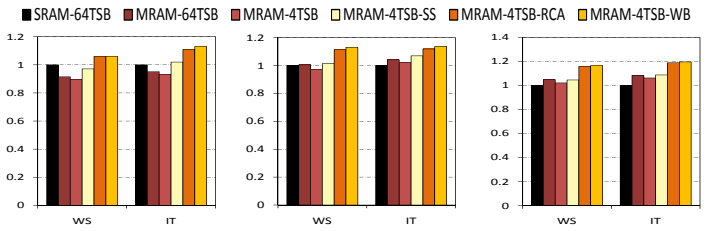
Energy saving: Figure 8 shows the savings in un-core (interconnect and cache) energy with our three proposed schemes. The energy results are shown to indicate that our proposal has no adverse impact on energy. The energy numbers are normalized to the energy of our baseline design (SRAM-64TSB). Energy is reduced solely due to the use of low leakage energy STT-RAM cells and all of our three schemes show similar energy savings. On an average, we obtain 54% reduction in energy across all workloads with the STT-RAM-4TSB-WB scheme. Reduction in energy consumption depends on the write intensity of the applications - applications with high write intensity to L2 show less reduction in energy compared to less write intensive applications.

Performance analysis with multi-programmed workloads: Next, we evaluate the effectiveness of our schemes using three diverse multi-programmed workload combinations:

- **Case-1:** We mix 16 copies each of 4 write intensive applications (`soplex`, `cactus`, `lbm` and `hmmer`) - a worst case scenario when SRAM banks are replaced by STT-RAM banks.
- **Case-2:** It consists of 16 copies each of 2 bursty and write intensive (`lbm` and `hmmer`), and 2 read intensive (`bzip` and `libqntm`) application mixes.
- **Case-3:** It is a realistic case consisting of an aggregate of 32 workload mixes, where each workload has a mix of 8 applications (8 copies each); each of the 32 workload mixes are spread out across the design space: 8 workloads represent read intensive applications, 8 workloads represent write intensive applications, and 16 workloads represent combination of read, write, and compute intensive applications. Within each of these three categories, applications are *randomly picked* to form the mixes.

Figure 9 shows the weighted speedup and instruction throughput improvement of our scheme across the three cases compared to the baseline. Observations from these evaluations are:

- In Case-1, when all write intensive applications are co-scheduled on the same CMP, there are no speedup improvements when SRAM is simply replaced by STT-RAM. On the other hand, our proposed WB scheme by selectively prioritizing only those packets for which *useful* work gets done, helps achieve 6% and 13% improvement in weighted speedup and instruction throughput, respectively. This case study highlights the importance of our proposal: in a worst case but quite probable scenario, the long write latency of STT-RAM cells can degrade performance by as much as 9% (weighted speedup with STT-RAM-64TSB). However, intelligently scheduling requests on to the banks using the NoC not only amortizes the performance degradation, but also can lead to performance improvement (7% weighted speedup with STT-RAM-4TSB-WB).
- In Case-2, when write+bursty and read intensive applications are co-scheduled, the performance improvements with our schemes are again better. However, an important observation made in this case study is that: when SRAM is replaced by STT-RAM, the bursty applications hog the network resources and consequently the read intensive applications are unfairly slowed down. This is highlighted in Figure 10, where we plot the maximum slowdown experienced by each application in the workload (smaller maximum slowdown is better) for the STT-RAM-64TSB and STT-RAM-4TSB-WB schemes. We find that the slowdown experienced by the read intensive applications (`bzip` and `libqntm`) is similar to what the write-intensive applications experience despite the fact that the read intensive benchmarks have lower miss-rates than the write intensive applications. This is because, since the write intensive applications in this workload are bursty and owing to longer write latency of their request packets, the read in-



(a) Case 1 (b) Case 2 (c) Aggregate across 32 cases

Figure 9: Weighted speedup (WS) and instruction throughput (IT) for the multi-programmed workloads.

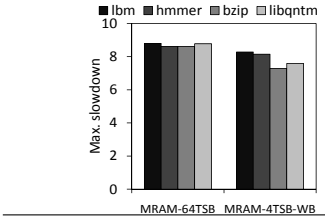
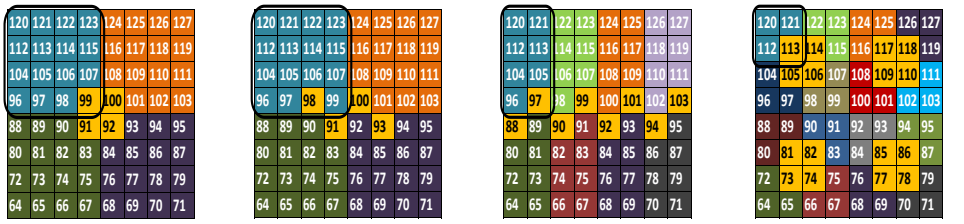


Figure 10: Maximum slowdown of the applications in Case-2.



(a) 4 regions, TSBs in corner (b) 4 regions, TSBs staggered (c) 8 regions, TSBs staggered (d) 16 regions, TSBs in corner

Figure 11: Illustration showing the various placement of TSBs and the sub-division of cache layer into regions. The yellow cells denote TSBs and the rest of the colored cells denote STT-RAM cache banks.

tensive applications' packets are not granted network resources by an un-oblivious STT-RAM router. However, with the WB scheme, when the destination STT banks are busy serving writes, the read packets from read intensive applications are prioritized over write packets and this decreases their maximum slowdown by 14%. Thus, the proposed schemes apart from improving performance, also aid in establishing a certain level of *fairness* across applications co-scheduled in the CMP.

- Finally, in Case-3, the aggregate results obtained are consistent with the previous performance numbers. When averaged across 32 multi-programmed workloads, the WB scheme improves system throughput by 16% and instruction throughput by 19% compared to the SRAM based cache design.

In summary, from among the six design alternatives, the WB scheme provides the best system performance and energy efficiency over a wide variety multi-threaded and multi-programmed workloads. In addition, it also introduces a level of fairness among the applications which neither SRAM nor STT-RAM banks alone can provide.

4.3 Sensitivity analysis

Sensitivity to the placement and the number of TSBs: All of our prior evaluations considered the cache layer to be partitioned into 4 regions, with 4 TSBs to route requests from the core layer to the cache layer. Additionally, the TSBs were placed at the corner of each region. In this section, we analyze the sensitivity of our proposals to the number and placement of TSBs, and sub-division of the cache layer into more regions. For this sensitivity study, we first start by experimenting with the placement of core to cache TSBs. Figure 11 shows the two different placements that we experimented with. In the *corner* placement of TSBs, all the 4 core to cache layer TSBs are placed at the corner of the regions, while in the *stagger* placement, the TSB placements are staggered. The advantage of staggering is that, when using X-Y routing in the core layer, the Y-direction flows going to the TSBs do not overlap with each other because the TSBs are now placed along different columns. Additionally, to analyze the sensitivity of our scheme to the number of regions, we sub-divided the cache layer into 4, 8 and 16 logical divisions.

Figure 12 shows the results of this analysis. All the performance numbers in this plot are averaged across all of the 42 applications and are normalized to the performance number with 4 regions and corner placement of TSBs in the regions. We find that, on an average, there is 3% improvement in IPC with the staggering placement of TSBs compared to the corner placement. Additionally, we find that subdivision of cache regions into many sub-regions hurts performance. This is because, as the number of regions grows, the number of cache banks in a region reduces and the opportunity for delaying requests to banks also reduces. With 16 regions for a 64 node cache layer, each region has only 4 banks and the distance between a parent and child nodes is reduced to 1 hop, thereby reducing the number of requests that can be re-ordered. We observe that the staggered placement of TSBs over 8 regions results in the best performance, resulting on an average, 5% IPC improvement compared to the corner placement of TSBs in 4 regions. With 16 regions, there is 10% IPC degradation compared to the corner placement of TSBs in 4 regions.

Sensitivity to distance between parent and child nodes: For this analysis, we experimented with the sensitivity of prioritizing requests from parent routers H hops away from the child nodes. Figure 13 (a) shows the number of potential requests in a parent node H-hop away from the child node when H varies from 1-3 hops across a few applications from our benchmark suite. We find that, if a parent node is too close to child nodes, then the opportunity for re-ordering reduces since the potential number of requests that can be re-ordered is few. When H is 3, each parent node has four child nodes, and hence, the potential number of requests that can be re-ordered increases. However, if re-ordering is done from three hops away (or farther) routers, then congestion estimation is not very accurate and this affects the delay of individual requests to busy child nodes (shown in Figure 13 (b)). After an extensive sensitivity analysis, we found that for $H = 2$, the number of requests to re-order

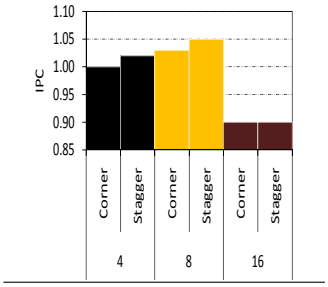


Figure 12: Sensitivity to placement of TSBs and number of cache regions.

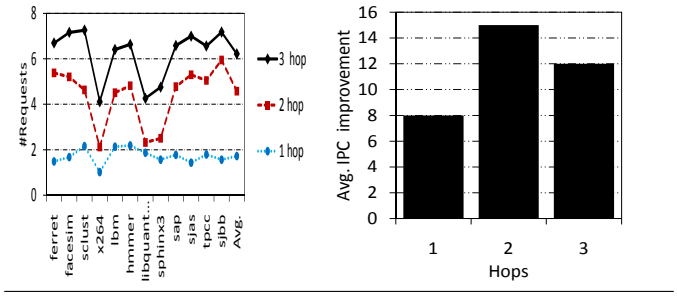


Figure 13: Average number of request-packets in a router in the cache layer to 1 hop, 2 hops and 3 hops away STT-RAM destination and sensitivity to hop distance.

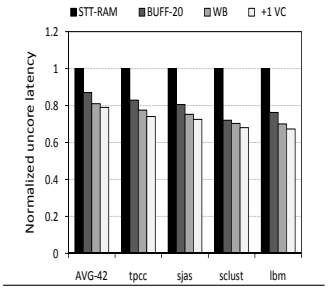


Figure 14: Latency reduction comparison with write-buffering (normalized to STT-RAM with no write-buffering).

is ideal for accurate congestion estimation, and all our results are thus based on two hops congestion calculation.

4.4 Comparison with an SRAM write buffer scheme

A previous work by Sun et al. in [19] evaluated the benefits of adding a small SRAM write buffer to *each* STT-RAM bank for hiding long latency writes. In their implementation, the cache bank controller services a write request by writing it into an SRAM buffer and when the bank is idle, the writes are written back to the STT-RAM bank. On a read request, the address is searched in parallel in the write-buffer and also in the STT-RAM bank for a hit. The authors found a 20 entry write-buffer as the optimal SRAM buffer sizes for STT-RAM banks. A large entry write buffer increases the design complexity as well as the area overhead. Additionally, the larger the buffer is, the longer it takes to check whether there is a hit in the buffer and then to access it. Large entry write buffers prove beneficial only when a benchmark is bursty. For comparison purposes, we analyzed the benefits of our WB scheme with a 20-entry write buffer for a few bursty and write intensive benchmarks. Figure 14 depicts the results of this comparison, where we show the normalized latency of the un-core part of processor requests i.e. the round-trip latency from a core to a cache bank through the network and back to the core. Observations from this analysis follow in order:

- The un-core latency of our scheme (annotated as WB in the figure) is, on an average, 6% better than that with STT-RAM banks having a 20-entry write buffer (annotated BUFF-20 in the figure). With the write-buffering scheme, the 1-cycle overhead in detecting if a request is a read/write before being written into the write-buffer is in the critical path of every request.
- A write-buffer is not beneficial when the benchmark is read intensive and/or is not bursty. In such a case, the write-buffer contributes to area and latency (for 1-cycle detection) overhead and there is negligible performance improvement. In fact, when averaged across 42 applications (annotated as AVG-42 in the figure), BUFF-20 only reduces un-core latency by 12.5% (only 2.25% with read intensive benchmarks), whereas the WB scheme reduces latency by 18.5% (up to 5.5% with read intensive benchmarks). Moreover, for bursty and write intensive benchmarks (e.g. tpcc, sjas, sclust and lbm), the WB scheme is 6% better than BUFF-20 since the WB scheme does a better job of managing the writes and burstiness in the network.
- Coupled with the write-buffering scheme, Sun at al. proposed a read-preemption scheme that can preempt a long-latency write request. In this case, an un-finished write-request is stored in an additional buffer. We find that, with bursty applications, preemption occurs more frequently than desired, thereby becoming an overhead.
- Instead of having a 20 entry write-buffer for every STT-RAM bank, if we increase the number of VCs in the routers by one (result annotated as +1 VC in the figure), there is an opportunity for an additional 1.6% latency reduction (compared to the WB scheme) with a 97% area reduction over the BUFF-20 scheme. This is because more flows are benefited when resources in the network are increased as opposed to increasing resources at the end-points.
- Buffering at the end points reduces the possibility of providing application level fairness which our WB scheme can provide as shown earlier.

In conclusion, we believe, buffering request intelligently in the network is more beneficial (and practical to implement since on-chip routers are already equipped with buffers) than buffering them at the end-points.

5. RELATED WORK

Dong et al. [4] studied the STT-RAM circuit design and presented a performance, power, and area model for STT-RAM caches and analyzed the benefits of 3D-stacked STT-RAMs at architectural level. Sun et al. [19] tackled the STT-RAM write problem by using a read-preemptive write buffer technique that allows read operations to terminate ongoing write operations under certain conditions. Zhou et al. [26] discussed the details of circuit design for STT-RAM early write termination to reduce STT-RAM write overhead. Our work is architecturally different from Zhou et al.'s work in the sense that we try to leverage an on-chip network to handle the write problem and is simpler than introducing additional gates for detection and termination of writes inside each STT-RAM sub-banks as proposed by them. Additionally, our proposal can complement the read-preemption scheme proposed in [19] by having the *network* prioritize reads over writes to the same bank rather than having the bank controller perform this prioritization. Overall, none of the previous works have leveraged the NoC design to mitigate the longer write delay problem in STT-RAMs.

6. CONCLUSIONS

STT-RAM possesses many attractive characteristics such as fast access time and low standby power. Alongside, the emerging 3D integration technology provides a cost-efficient way to integrate STT-RAM with multi-core architectures/CMPs. However, one of the systemic disadvantages of STT-RAM technology is the latency

associated with the write operations. In this work, we have proposed an elegant network level solution to alleviate this problem. Our proposed network level solution centers around prioritizing packets to idle banks and delaying accesses to an STT-RAM bank currently servicing a write request. Our study shows that by accurately estimating the busy status of a cache bank two hops away using the WB estimation, we can prioritize requests to idle banks for effectively hiding the long write latency.

Experimental results using a 128-node (64 core and 64 cache banks) CMP system show that the proposed on-chip network solution can lead to an average 14% improvement in IPC and 19% improvement in instruction throughput across a diverse set of 42 applications including both multi-threaded and multi-programmed workloads. Another important conclusion of this study is that the proposed network tailored solution is more efficient (in terms of performance and resource overheads) compared to a recently proposed write-buffer mechanism for hiding write latency. Overall, we believe that our proposal is promising to improve the performance and power envelope of STT-RAM based stacked CMP architectures.

7. ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their reviews and comments in improving this paper. This work is supported in part by National Science Foundation (NSF) grants CCF-0702617, CNS-0916887, CCF-0903432, CNS-0905365, and CCF-1017277, SRC grant, and by DoE's ASCR program.

8. REFERENCES

[1] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. H. Loh, D. McCaule, P. Morrow, D. W. Nelson, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J. Shen, and C. Webb. Die Stacking (3D) Microarchitecture. In *MICRO-39*, 2006.

[2] W. J. Dally and B. Towles. Route Packets, Not Wires: On-Chip Interconnection Networks. In *38th DAC*, 2001.

[3] R. Das, S. Eachempati, A. Mishra, V. Narayanan, and C. Das. Design and Evaluation of a Hierarchical On-Chip Interconnect for Next-Generation CMPs. In *15th HPCA*, 2009.

[4] X. Dong, X. Wu, G. Sun, Y. Xie, H. Li, and Y. Chen. Circuit and Microarchitecture Evaluation of 3D Stacking Magnetic RAM (MRAM) as a Universal Memory Replacement. In *45th DAC*, 2008.

[5] S. Eyerman and L. Eeckhout. System-Level Performance Metrics for Multiprogram Workloads. *IEEE Micro*, 2008.

[6] P. Gratz, B. Grot, and S. Keckler. Regional Congestion Awareness for Load Balance in Networks-on-Chip. In *14th HPCA*, 2008.

[7] X. Guo, E. Ipek, and T. Soyata. Resistive Computation: Avoiding the Power Wall with Low-Leakage, STT-MRAM Based Computing. In *37th ISCA*, 2010.

[8] M. Hosomi, H. Y. Yamagishi, T. Yamamoto, K. Bessho, Y. Higo, K. Yamane, H. Yamada, M. Shoji, H. Hachino, C. Fukumoto, H. Nagao, and H. Kano. A Novel Nonvolatile Memory with Spin Torque Transfer Magnetization Switching: Spin-RAM. In *IEDM*, 2005.

[9] Y. Joo, D. Niu, X. Dong, G. Sun, N. Chang, and Y. Xie. Energy and Endurance-Aware Design of Phase Change Memory Caches. In *DATE*, 2010.

[10] T. Kawahara, R. Takemura, K. Miura, J. Hayakawa, S. Ikeda, Y. Lee, R. Sasaki, Y. Goto, K. Ito, I. Meguro, F. Matsukura, H. Takahashi, H. Matsuoka, and H. Ohno. 2Mb Spin-Transfer Torque RAM (SPRAM) with Bit-by-Bit Bidirectional Current Write and Parallelizing-Direction Current Read. In *ISSCC*, 2007.

[11] T. Kgil, S. D'Souza, A. Saidi, N. Binkert, R. Dreslinski, T. Mudge, S. Reinhardt, and K. Flautner. PicoServer: Using 3D Stacking Technology to Enable a Compact Energy Efficient Chip Multiprocessor. *ASPLOS-XII*, 2006.

[12] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting Phase Change Memory as a Scalable DRAM Alternative. In *36th ISCA*, 2009.

[13] N. Madan, L. Zhao, N. Muralimanohar, A. Udipi, R. Balasubramonian, R. Iyer, S. Makineni, and D. Newell. Optimizing Communication and Capacity in a 3D Stacked Reconfigurable Cache Hierarchy. In *15th HPCA*, 2009.

[14] N. Muralimanohar, R. Balasubramonian, and N. Jouppi. Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0. In *MICRO-40*, 2007.

[15] L.-S. Peh and W. J. Dally. A Delay Model and Speculative Architecture for Pipelined Routers. In *7th HPCA*, 2001.

[16] M. K. Qureshi, J. P. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali. Enhancing Lifetime and Security of PCM-Based Main Memory with Start-Gap Wear Leveling. In *MICRO-42*, 2009.

[17] M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable High Performance Main Memory System Using Phase-Change Memory Technology. In *36th ISCA*, 2009.

[18] A. Snavely and D. M. Tullsen. Symbiotic Jobscheduling for a Simultaneous Multithreaded Processor. In *ASPLOS-IX*, 2000.

[19] G. Sun, X. Dong, Y. Xie, J. Li, and Y. Chen. A Novel Architecture of the 3D Stacked MRAM L2 Cache for CMPs. In *15th HPCA*, 2009.

[20] M. Tremblay and S. Chaudhry. A Third-Generation 65nm 16-Core 32-Thread Plus 32-Scout-Thread CMT SPARC Processor. In *ISSCC*, 2008.

[21] H. Wang, X. Zhu, L.-S. Peh, and S. Malik. Orion: A Power-Performance Simulator for Interconnection Networks. In *MICRO-35*, 2002.

[22] D. H. Woo, N. H. Seong, D. L. Lewis, and H.-H. S. Lee. An Optimized 3D-Stacked Memory Architecture by Exploiting Excessive, High-Density TSV Bandwidth. In *16th HPCA*, 2010.

[23] Y. Xie. Modeling, Architecture, and Applications for Emerging Memory Technologies. *IEEE Design and Test of Computers, Special Issues on Memory Technologies*, 2010.

[24] Y. Xie, G. H. Loh, B. Black, and K. Bernstein. Design Space Exploration for 3D Architectures. *ACM JETC*, 2(2), 2006.

[25] W. Zhao, E. Belhaire, Q. Mistral, C. Chappert, V. Javerliac, B. Dieny, and E. Nicolle. Macro-Model of Spin-Transfer Torque Based Magnetic Tunnel Junction Device for Hybrid Magnetic-CMOS Design. In *BMAS*, 2006.

[26] P. Zhou, B. Zhao, J. Yang, and Y. Zhang. Energy Reduction for STT-RAM Using Early Write Termination. In *ICCAD*, 2009.